Practical Work Part 1: Basics of display and interaction

The exercises in this section cover the main concepts of event programming. They are to be done throughout the introductory course. **duration with course: 4h**

Objectives:

- Assemble nodes (shapes and widgets) in a simple layout,
- Set up transformations on nodes in the scene graph,
- Set up a data binding,
- Use a simple data model,
- React to a value change,
- React to an event all along its propagation chain.

Documentation:

JavaFX User Interface components
http://docs.oracle.com/javase/8/javafx/user-interface-tutorial/index.html

JavaFX Layouts
http://docs.oracle.com/javase/8/javafx/layout-tutorial/index.html

JavaFX Events processing
http://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

JavaFX Javadoc
https://docs.oracle.com/javase/8/javafx/api/toc.htm

Exercise 1: Scene graph, nodes and coordinate systems

1.1 Launch *NetBeans* and create a new project named "PracticalWork1" without main class (uncheck the box "Create Application Class"). Copy then Paste (Paste > Copy contextual menu) in its "Source Packages" folder the `Ex1_CoordinateSystems` class provided with this subject on e-campus.

This Application class sets up nested containers with absolute layouts, i.e. placing their children explicitly in (x, y) in their own coordinate system. A rectangle is placed inside the lowest container in the scene graph. The code allows to listen mouse clicks on the rectangle (we'll see how to listen mouse clicks later, just use the code as it is for now) and, when clicked, to display mouse coordinates in the rectangle coordinate system. Read the code and draw the scene graph to show the nesting of components and their coordinate systems. Check your understanding by launching the application and clicking in the corners of the rectangle (the values in the coordinate system of the rectangle are easy to imagine).

1.2 Transform the code so that it also displays the mouse coordinates in the parent of the rectangle, the scene, and the stage coordinate systems. To do so you will use the conversion methods presented in the course (it's just to practice, we will see later with the events that it's not always the simplest way to do it).

Exercise 2: Transformations and coordinate systems

2.1 Copy then Paste (Paste > Refactor Copy contextual menu) the `Ex1_CoordinateSystems` class of the first exercise. That option will give you the opportunity to rename the new class, type `Ex2_CoordinateSystems`. In this new class increase the scale of the nested container by a factor 2. To do so you will add a `Scale` instance to the nested container's transforms list.

2.2 What is the result of this modification on the displayed coordinates when you click in the corners of the rectangle?

Exercise 3: Data binding

One of *JavaFX*'s event-driven mechanisms, called data binding, allows you to bind two properties, unidirectionally or bidirectionally, ensuring that when one of the values changes, it will be copied to the other. The linked properties cannot be of a primitive type, they must be of a subtype of the `Property` class (Properties simply encapsulate the primitive types and provide the binding mechanism).

3.1 Identify in the `Property` class javadoc (or in the course) the two methods for setting up a unidirectional and bidirectional data binding.

3.2 Create a new *JavaFX* main class (named `Ex3_Bindings`) whose root of the scene graph will be a container that can align its children horizontally (look for such a container in one of the sources of the documentation).
In this container you will place two instances of the `Slider` class and bind the value of the second instance to the value of the first instance (unidirectional binding).

What happens when you handle the first slider? Same question for the second one.

3.3 Add an instance of the `TextField` class between the two sliders and bind its content to the value of the first slider.
What's going on? What kind of mechanism would need to be put in place to make it work?
Look for a concrete solution in the javadoc of the `NumberExpressionBase` class, from which all properties encapsulating a number inherit.

3.4 Make the data binding between the two sliders bidirectional.

3.5 Try to do the same between the content of the text field and the value of the first slider. You are facing the same problem of data type conversion except that this time the conversion must be done in both directions. Find the solution in the `StringProperty` doc (the type of data contained in a text field).

Exercise 4: Data model

Sliders and text allow to display but also to modify a value. Sliders and text will therefore be referred to as ***view/controller*** items and the value will be called a ***data model***. What we did in the previous exercise has the drawback of mixing the *view/controller* set and the *data model*, which can quickly make the code difficult to maintain in a larger application.

Create a copy of `Ex3_Bindings` class, named `Ex4_DataModel`. In order to organize the whole thing properly, isolate the model in a suitable `Property` (encapsulating a `Double`) and modify the bindings so that they are no longer made between the different widgets but directly to the *model*.

In bigger applications *data models* and complex computations are externalized in specific classes constituting what is called a ***functional core***, so that they are completely separated from their associated *views/controllers.*

Exercise 5: React to a value change

For more complex reactions to changes in the value of a `Property` (or if it is necessary to create edge effects), we cannot use `Bindings`. We now need to invoke a method in order to compute something each time the value changes. To do so we use a `ChangeListener` and a subscription mechanism.

5.1 Create a copy of `Ex4_DataModel` class, named `Ex5_ChangeListener`. In this new class add a `ChangeListener` that allows to print an alert message in the console when the *data model* exceeds a threshold of 75. As there are classes named `ChangeListener` in several libraries pay attention when you type it to choose the right one among the different alternatives *NetBeans* proposes to import (coming from a *JavaFX* package, not a *Swing* or another package). From now on you will always have to pay attention to this kind of detail for each import.

What happens when you move the slider repeatedly over the threshold?

5.2 We now want a single alert when the threshold is exceeded and an end of alert when the value falls below the threshold again. And each message must be dated.
The date can be obtained from the system and formatted into a string with the following instruction:
`String date = new SimpleDateFormat("HH:mm:ss.SSS").format(new Date());`

5.3 Finally, as writing in the console is not what we can call a graphical user interface you will also change the color of the text field to red when in alert, and back to black when not in alert.
You can change the color by setting a CSS style on the text field (the method: `setStyle(...)`, the style: `-fx-text-fill`).

Exercise 6: React to an event all along its propagation chain

In many cases it is not enough just to react to a value change. We now need to invoke a method in order to compute something each time a specific event occurs, for example when a button is pressed. To do so we use an `EventListener` and 2 subscription mechanisms, one of them being very similar to those of the `ChangeListener`.

6.1 Copy then Paste (Paste > Copy contextual menu) the `Ex6_Events` class provided with this subject. It is based on the Helloworld we have examined in course. You can see that the code allows to react to `ActionEvent` fired on the button. The subscription is done by using the convenience method, which is the simplest method. But in this exercise we will study the propagation chain of events, so we cannot use convenience methods. Use instead the full subscription with the option (filter or handler) at your convenience.

6.2 Now subscribe for the same event type on the root container and choose the correct option to ensure to react before those on the button.

6.3 Finally subscribe for the same event type on the root container but this time the subscription must ensure to react after those on the button.

6.4 Now what happens if the instruction `event.consume()` is added at the end of the change method from the question 6.2?

Congratulations! At the end of this part you have seen the main concepts that will allow you to design an interactive application and develop it in *JavaFX*.

From now on we will build more complex applications using the *event-driven programming* concepts, architecting the code in several classes and separating the *model* and *view/controller* aspects.

See you soon!